

**DEVICE AND METHOD FOR SCHEDULING AND STORAGE MEDIUM**

Patent Number: JP2000020323  
Publication date: 2000-01-21  
Inventor(s): MURATA SEIJI  
Applicant(s): SONY CORP  
Requested Patent: JP2000020323  
Application Number: JP19990068958 19990315  
Priority Number(s):  
IPC Classification: G06F9/46  
EC Classification:  
Equivalents:

---

**Abstract**

---

**PROBLEM TO BE SOLVED:** To provide a method, in which the problem of priority inversion can be avoided, the change of queue is reduced and further the processing time of a CPU can be effectively utilized, as a scheduling method in a multi- thread system for performing time division processing to plural threads.

**SOLUTION:** Time slot data A, B and C are allocated to the first queue corresponding to respective threads, the scheduling is performed on each of the time slot data as one unit and when the processing time is allocated to each of the time slot data, the thread corresponding to the relevant one of the time slot data is executed. When it is necessary for the thread having the high priority to wait the processing of low-priority thread, until the relevant processing is completed, the time slot data corresponding to the thread having the high priority are handled as the time slot data corresponding to the thread having the low priority. When the processing times are allocated to these relevant time slot data, the thread having the low priority is executed.

---

Data supplied from the **esp@cenet** database - I2



(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号

特開2000-20323

(P2000-20323A)

(43) 公開日 平成12年1月21日 (2000.1.21)

(51) Int.Cl.<sup>7</sup>

G 0 6 F 9/46

識別記号

3 4 0

F I

G 0 6 F 9/46

テーマコード\* (参考)

3 4 0 B

3 4 0 E

審査請求 未請求 請求項の数12 O L (全 22 頁)

(21) 出願番号 特願平11-68958

(22) 出願日 平成11年3月15日 (1999.3.15)

(31) 優先権主張番号 特願平10-117504

(32) 優先日 平成10年4月27日 (1998.4.27)

(33) 優先権主張国 日本 (J P)

(71) 出願人 000002185

ソニー株式会社

東京都品川区北品川6丁目7番35号

(72) 発明者 村田 誠二

東京都品川区北品川6丁目7番35号 ソニ

ー株式会社内

(74) 代理人 100067736

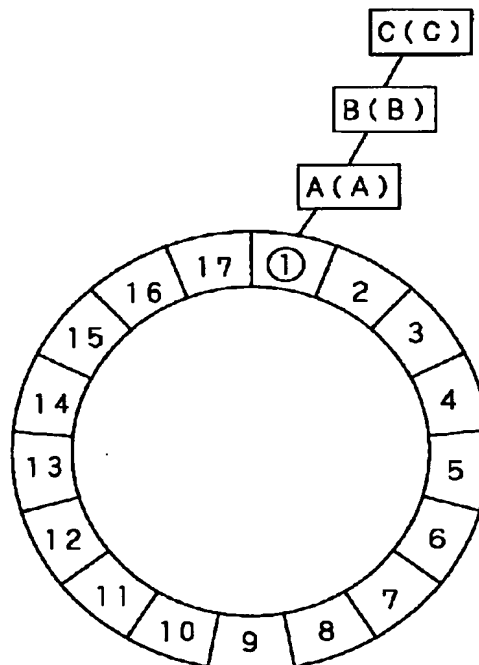
弁理士 小池 晃 (外2名)

(54) 【発明の名称】 スケジューリング装置及び方法並びに記録媒体

(57) 【要約】

【課題】 複数のスレッドを時分割処理するマルチスレッドシステムにおけるスケジューリング方法として、優先度逆転問題を回避することができ、待ち行列の変更が少なく済み、しかも、CPUの処理時間を有効に利用することが可能な方法を提供する。

【解決手段】 各スレッドに対応するようにタイムスロットデータを割り当て、タイムスロットデータを単位としてスケジューリングを行い、タイムスロットデータに処理時間が割り当てられたら、当該タイムスロットデータに対応したスレッドを実行する。そして、優先度の高いスレッドが優先度の低いスレッドの処理を待つ必要がある場合には、当該処理が完了するまでの間、優先度の高いスレッドに対応していたタイムスロットデータを、優先度の低いスレッドに対応したタイムスロットデータとして扱い、当該タイムスロットデータに処理時間が割り当てられたら、優先度の低いスレッドを実行する。



【特許請求の範囲】

【請求項1】 複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング装置において、  
各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータを記憶する手段と、  
タイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うスケジューラと、  
処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行する手段と、  
優先度の高い第一の実行主体が、優先度の低い第二の実行主体の処理を待つ必要があるときには、当該第一の実行主体に対応するタイムスロットデータを当該第二の実行主体に割り当てる手段と、  
当該第二の実行主体の処理が終了したときに、当該第一の実行主体から割り当てられたタイムスロットデータを、当該第一の実行主体に再度割り当てる手段と、  
を具備することを特徴とするスケジューリング装置。  
【請求項2】 前記タイムスロットデータは、各タイムスロットデータの対応する実行主体を指し示すリンクデータを格納するデータ領域を具備し、  
前記スケジューリング装置は、  
優先度の高い実行主体に対応していたタイムスロットデータを、優先度の低い実行主体に対応したタイムスロットデータとして扱うようにするときには、当該タイムスロットデータのリンクデータが指し示す実行主体を、優先度の高い実行主体から優先度の低い実行主体に変更する手段を具備する、  
ことを特徴とする請求項1記載のスケジューリング装置。  
【請求項3】 複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング装置において、  
各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータを記憶する手段と、  
タイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うスケジューラと、  
少なくとも一つのタイムスロットデータを配置できる複数の待ち行列から構成されるリングバッファと、  
処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行する手段と、  
タイムスロットデータに対応する実行主体の優先度に応じて、当該タイムスロットデータを配置すべきリングバッファ上の待ち行列を決定する位置決定手段と、  
タイムスロットデータを前記リングバッファ上の待ち行列に配置するときに、当該リングバッファ上の待ち行列の最後尾に当該タイムスロットデータを配置する手段と、  
所定のリングバッファ上の待ち行列の先頭に位置するタイムスロットデータに処理時間を与える手段と、

を具備することを特徴とするスケジューリング装置。

【請求項4】 前記タイムスロットデータは、タイムスロットデータ毎に対応する実行主体の優先度を保持するデータ領域を具備し、

前記位置決定手段は、

割り当てられた処理時間の枠を使い果たした実行主体に対応するタイムスロットデータをリングバッファ上の待ち行列に配置するとき、当該リングバッファ上の待ち行列の数から1を引いた数と当該タイムスロットデータの保持する優先度の差分だけ、当該リングバッファ上で当該タイムスロットデータが占める現在の待ち行列の位置から移動した位置の待ち行列に配置することを決定する、

ことを特徴とする請求項3記載のスケジューリング装置。

【請求項5】 複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング方法において、

各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、

処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、

優先度の高い第一の実行主体が、優先度の低い第二の実行主体の処理を待つ必要があるときには、当該第一の実行主体に対応するタイムスロットデータを当該第二の実行主体に割り当てるステップと、

当該第二の実行主体の処理が終了したときに、当該第一の実行主体から割り当てられたタイムスロットデータを、当該第一の実行主体に再度割り当てるステップと、  
を具備することを特徴とするスケジューリング方法。

【請求項6】 前記タイムスロットデータ毎に、対応する実行主体を指し示すリンクデータが格納されるデータ領域を確保しておく、

優先度の高い実行主体に対応していたタイムスロットデータを、優先度の低い実行主体に対応したタイムスロットデータとして扱うようにするときには、当該タイムスロットデータに対応したリンクデータが指し示す実行主体を、優先度の高い実行主体から優先度の低い実行主体に変更する、

ことを特徴とする請求項5記載のスケジューリング方法。

【請求項7】 複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング方法において、

各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、

処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、

タイムスロットデータに対応する実行主体の優先度に応じて、リングバッファ上の複数の待ち行列から当該タイムスロットデータを配置すべき待ち行列を決定する位置決定ステップと、  
タイムスロットデータを前記リングバッファ上の待ち行列に配置するときに、当該リングバッファ上の待ち行列の最後尾に当該タイムスロットデータを配置するステップと、  
所定のリングバッファ上の待ち行列の先頭に位置するタイムスロットデータに処理時間を与えるステップと、  
を具備することを特徴とするスケジューリング方法。

【請求項8】 前記位置決定ステップは、  
対応する実行主体の優先度を保持する領域を前記タイムスロットデータ毎に確保しておき、割り当てられた処理時間の枠を使い果たした実行主体に対応するタイムスロットデータをリングバッファ上の待ち行列に配置するとき、当該リングバッファ上の待ち行列の数から1を引いた数と当該タイムスロットデータの保持する優先度の差分だけ、当該リングバッファ上で当該タイムスロットデータが占める現在の待ち行列の位置から移動した位置の待ち行列に配置することを決定する、  
ことを特徴とする請求項7記載のスケジューリング方法。

【請求項9】 複数の実行主体を時分割処理することが可能なオペレーティングシステムのデータ処理プログラムが記録された記録媒体であって、  
各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、  
処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、  
優先度の高い第一の実行主体が、優先度の低い第二の実行主体の処理を待つ必要があるときには、当該第一の実行主体に対応するタイムスロットデータを当該第二の実行主体に割り当てるステップと、  
当該第二の実行主体の処理が終了したときに、当該第一の実行主体から割り当てられたタイムスロットデータを、当該第一の実行主体に再度割り当てるステップと、  
を処理するデータ処理プログラム、  
を提供することを特徴とする記録媒体。

【請求項10】 上記データ処理プログラムは、  
前記タイムスロットデータ毎に、対応する実行主体を指し示すリンクデータが格納されるデータ領域を確保しておき、  
優先度の高い実行主体に対応していたタイムスロットデータを、優先度の低い実行主体に対応したタイムスロットデータとして扱うようにするときには、当該タイムスロットデータに対応したリンクデータが指し示す実行主体を、優先度の高い実行主体から優先度の低い実行主体に変更する、

ことを特徴とする請求項9記載の記録媒体。

【請求項11】 複数の実行主体を時分割処理することが可能なオペレーティングシステムのデータ処理プログラムが記録された記録媒体であって、  
各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、  
処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、  
タイムスロットデータに対応する実行主体の優先度に応じて、リングバッファ上の複数の待ち行列から当該タイムスロットデータを配置すべき待ち行列を決定する位置決定ステップと、  
タイムスロットデータを前記リングバッファ上の待ち行列の一つに配置するときに、当該リングバッファ上の待ち行列の最後尾に当該タイムスロットデータを配置するステップと、  
所定のリングバッファ上の待ち行列の先頭に位置するタイムスロットデータに処理時間を与えるステップと、  
を処理するデータ処理プログラム、  
を提供することを特徴とする記録媒体。

【請求項12】 前記位置決定ステップは、  
対応する実行主体の優先度を保持する領域を前記タイムスロットデータ毎に確保しておき、割り当てられた処理時間の枠を使い果たした実行主体に対応するタイムスロットデータをリングバッファ上に配置するとき、当該リングバッファ上の待ち行列の数から1を引いた数と当該タイムスロットデータの保持する優先度の差分だけ、当該リングバッファ上で当該タイムスロットデータが占める現在の待ち行列の位置から移動した位置の待ち行列に配置することを決定する、  
ことを特徴とする請求項11記載の記録媒体。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、複数の実行主体を時分割処理することが可能なオペレーティングシステムにおけるスケジューリング装置及びスケジューリング方法、並びに、複数の実行主体を時分割処理することが可能なオペレーティングシステムが記録された記録媒体に関する。

【0002】

【従来の技術】オペレーティングシステム、並びにオペレーティングシステム上で動作するアプリケーションプログラム等における実行実体を、時分割にて並行して動作させることが可能なシステムにおいて、実行主体は、実行主体毎に設定された優先度に基づいてスケジューリングされる。具体的には、通常、各実行主体は、優先度の高い実行主体ほど早く実行されるように、優先度に基づいて待ち行列に入れられ、待ち行列に入っている実行主体が、当該待ち行列の順番に従ってCPU (Central

Processing Unit) によって順次実行される。ここで、実行主体は、プロセス、タスク、スレッドなどと一般に呼ばれるものであり、本明細書では以後、代表してスレッドと称する。また、スレッドが実行主体であるシステムのことをマルチスレッドシステムと称する。

【0003】すなわち、マルチスレッドシステムにおいて、全スレッドは、基本的には各スレッドの生成時等に設定された優先度に基づいて、それらの順序関係が定義される。なお、この順序関係は必ずしも実行順序の関係だけを示すものとは限らず、例えば、いわゆるタイムスライススケジューリングでは、ある時間内に各スレッドに割り当てられるCPUの処理時間の割合として、優先度に基づく順序関係が定義される。また、場合によっては、システムの時間経過や各スレッドの処理時間等に応じて動的に、各スレッドの優先度に重み付けをすることもある。

【0004】ところで、優先度に基づくスケジューリングのアルゴリズムでは、共有資源に対するアクセスの排他制御において、優先度逆転と呼ばれる問題が生じる場合がある。以下、この優先度逆転について、図26を参照して説明する。

【0005】図26は、優先度逆転について、その典型的な状況を示す図であり、優先度の高いスレッドAがクリティカルセクションCS1に入ろうとした場合に起こりうる状況を示している。なお、クリティカルセクションCS1は、スレッドAとスレッドCが共通に使用する資源を含んでいるものとする。

【0006】図26に示すように、まず、スレッドCがクリティカルセクションCS1に入った後に、より高い優先度を持つスレッドAが動作可能状態になったとする。すると、スレッドCの実行が中断され、スレッドAの実行が開始される。しかし、このとき既にスレッドCがクリティカルセクションCS1に入っているため、スレッドCがクリティカルセクションCS1を抜けるまで、スレッドAは処理を待たねばならない。そこで、スレッドAは待ち状態となり、再びスレッドCの実行が開始される。

【0007】そして、このような状況のときに、スレッドCがクリティカルセクションAを抜ける前に、スレッドAとスレッドCのそれぞれの優先度の中間の優先度を持つスレッドBが動作可能状態となった場合に、優先度逆転が生じる。すなわち、より優先度の高いスレッドAが待ち状態であるにも関わらずに、より優先度の低いスレッドBの実行が開始されるという状況が生じる。優先度が中間のスレッドBが動作可能状態になると、優先度に従って、スレッドCの実行が中断され、スレッドBの実行が開始される。このとき、スレッドAは中断したままである。しかも、スレッドAとスレッドBの関係は優先度以外には定義されていないので、このような状況になると、スレッドAは、自分がどの程度の時間、処理が

中断されるかを見積もることも不可能となる。

【0008】このように、優先度に基づくスケジューリングのアルゴリズムでは、より高い優先度のスレッドが、より低い優先度のスレッドの実行を待つような優先度逆転と呼ばれる問題が生じる可能性がある。しかも、上述したように、優先度逆転が生じると、より高い優先度のスレッドの中断時間を予測することが不可能となる。このように、高い優先度のスレッドの中断時間が予測不可能に陥ってしまうのは、特に実時間システム（各処理がある一定時間内に終わらなければシステムが成り立たないようなシステム）において、致命的な問題である。

【0009】そこで、従来、このような優先度逆転の問題を回避する必要がある場合には、優先度継承機構を導入するようにしている。(Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. Technical report, Computer Science Department, Carnegie Mellon University. November 1987. Tech. Report CMU-CS-87-181.) 優先度継承機構とは、高い優先度を持つスレッドが、低い優先度を持つスレッドのために待ち状態になった場合に、高い優先度を持つスレッドの優先度を、低い優先度を持つスレッドに継承する機構である。

【0010】図26と同様な状況において、優先度継承機構を導入した場合のシナリオを図27に示す。優先度継承機構を導入した場合には、スレッドAが待ち状態となって再びスレッドCの実行が開始されるときに、スレッドCの優先度が、スレッドAと同じレベルの優先度に設定される。この結果、スレッドCがクリティカルセクションCS1から抜けるまでは、スレッドAより優先度が低いスレッド（例えば図中のスレッドB）によって、スレッドCの実行が中断されてしまうようなことはなくなる。

【0011】このように優先度継承機構を導入した場合、スレッドCは、クリティカルセクションCS1を抜けるまで、スレッドAの優先度でスケジューリングされる。したがって、スレッドAがクリティカルセクションCS1にアクセスするために待たされる時間は、スレッドCがクリティカルセクションCS1を必要とする時間以内に制限されることとなり、優先度の高いスレッドであるスレッドAの中断時間の予測が可能となる。

【0012】ところで、以上のような優先度継承機構を実際に実現するには、具体的には下記のような処理が行われる。なお、ここでは、上述したように、優先度が低いスレッドCがクリティカルセクションCS1を抜けるまで、優先度が高いスレッドAが処理を待たねばならないような場合を例に挙げる。

【0013】まず、スレッドCがクリティカルセクションCS1を抜けるまでスレッドAが処理を待たねばならないと判断されると、スレッドAの状態が待ち状態とさ

れ、当該スレッドAが待ち行列から取り除かれる。次に、スレッドCの優先度が一時的にスレッドAの優先度に変更され、変更された優先度に基づいて、スレッドCが待ち行列に入れ直される。そして、待ち行列に入っているスレッドはCPUによって順次実行されていき、スレッドCが選択された時点で当該スレッドCが実行される。なお、スレッドCの実行中に、スレッドCに割り当てられていた一定の処理時間を使い切った場合、当該スレッドCは、変更された優先度（すなわちスレッドAの優先度）に基づいて再び待ち行列に入れられる。

【0014】その後、スレッドAの待ち事象が解決した時点（すなわち、スレッドCがクリティカルセクションCS1を抜けた時点）で、スレッドCの優先度が元に戻され、元に戻された優先度に基づいて、スレッドCが待ち行列に入れ直されるとともに、スレッドAの状態が実行可能状態とされ、当該スレッドAが待ち行列に入れられる。

【0015】以上の処理により、図27に示したような優先度継承機構が実現され、スレッドCがクリティカルセクションCS1から抜けるまでは、スレッドAより優先度が低いスレッド（例えば図中のスレッドB）によって、スレッドCの実行が中断されてしまうようなことはなくなる。

【0016】

【発明が解決しようとする課題】ところで、従来のスケジューリング機構では、各スレッドが持つその時点の優先度に応じて、スレッドの待ち行列に入る位置を決定している。そして、上述のような優先度継承機構では、優先度が動的に頻繁に変更されるため、待ち行列の変更を頻繁に行う必要が生じる。そのため、上述のような優先度継承機構を導入すると、優先度の動的な変更に伴う待ち行列のリスケジューリングによるオーバーヘッドが大幅に増加してしまう。

【0017】すなわち、上述のような優先度継承機構は、優先度逆転の問題を回避するためには有効であるが、当該優先度継承機構を導入すると、優先度が頻繁に動的に変更されるため、スケジューリングによるオーバーヘッドが大幅に増加し、システム全体の性能が低下してしまうという問題が生じる。

【0018】また、優先度がスレッドの単なる実行順序を示すのではなくCPUの処理時間の割合を示すような場合、すなわち時分割スケジューリングを採用しているような場合には、待ち状態の原因となっているスレッド（上記の例ではスレッドC）の優先度を、待ち状態にあるスレッド（上記の例ではスレッドA）と同じレベルの優先度に設定するだけでは、これらのスレッドに割り当てられている処理時間を有効に利用することができない。

【0019】すなわち、例えば、上記の例のように、スレッドCの優先度を、スレッドAと同じレベルの優先度

に設定した場合、スレッドCはスレッドAの優先度にて実行されるため、スレッドAに割り当てられていたCPUの処理時間は使用されるが、スレッドCに割り当てられていたCPUの処理時間が使用されなくなる。そして、このような状態を回避して、CPUの処理時間を有効に使用するためには、待ち状態にあるスレッドAに割り当てられていた処理時間と、待ち状態の原因となっているスレッドCに割り当てられていた処理時間との合計を求めて、その処理時間をスレッドCに改めて割り当てるようにする必要がある。換言すれば、時分割スケジューリングを採用しているような場合には、スレッドAに割り当てられていた処理時間と、スレッドCに割り当てられていた処理時間との合計の処理時間が与えられる優先度を求めて、当該優先度をスレッドCに割り当てるようにすべきである。

【0020】しかしながら、このようにスレッドに割り当てられる処理時間を計算し直して、スケジューリングし直すようにすると、当該処理に伴うオーバーヘッドが大幅に増大してしまう。そのため、従来の優先度継承機構では、各スレッドに割り当てられたCPUの処理時間を有効に利用することができなかった。

【0021】本発明は、以上のような従来の実情に鑑みて提案されたものであり、優先度逆転問題を回避することが可能なスケジューリング装置及び方法を提供することを目的としている。本発明にあるようなスケジューリング装置及び方法を用いることにより、オーバーヘッドの増加の原因となる待ち行列の変更が少なくて済み、しかも、割り当てられたCPUの処理時間を有効に利用することが可能になる。また、本発明は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのデータ処理プログラムが記録された記録媒体を提供することも目的としている。

【0022】

【課題を解決するための手段】本発明に係るスケジューリング装置は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング装置であり、各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータを記憶する手段と、タイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うスケジューラと、処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行する手段と、優先度の高い第一の実行主体が、優先度の低い第二の実行主体の処理を待つ必要があるときには、当該第一の実行主体に対応するタイムスロットデータを当該第二の実行主体に割り当てる手段と、当該第二の実行主体の処理が終了したときに、当該第一の実行主体から割り当てられたタイムスロットデータを、当該第一の実行主体に再度割り当てる手段とを具備することを特徴とする。

【0023】以上のような本発明に係るスケジューリン

グ装置では、実行主体から独立したタイムスロットデータをスケジューリングの対象とすることにより、優先度の高い実行主体が、優先度の低い実行主体の処理を待つ必要のあるときには、タイムスロットデータを受け渡すという低コストな処理によって優先度継承を行うことができる。

【0024】また、本発明に係るスケジューリング装置は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング装置であり、各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータを記憶する手段と、タイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うスケジューラと、少なくとも一つのタイムスロットデータを配置できる複数の待ち行列から構成されるリングバッファと、処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行する手段と、タイムスロットデータに対応する実行主体の優先度に応じて、当該タイムスロットデータを配置すべきリングバッファ上の待ち行列を決定する位置決定手段と、タイムスロットデータを前記リングバッファ上の待ち行列に配置するときに、当該リングバッファ上の待ち行列の最後尾に当該タイムスロットデータを配置する手段と、所定のリングバッファ上の待ち行列の先頭に位置するタイムスロットデータに処理時間を与える手段とを具備することを特徴とする。

【0025】以上のような本発明に係るスケジューリング装置では、タイムスロットデータに対応する実行主体の優先度に応じて、当該タイムスロットデータを配置すべきリングバッファ上の位置を決定し、同じ優先度のタイムスロットデータがリングバッファ上で同じ位置を共有することにより、タイムスロットの優先度に応じて、タイムスロットデータに対応する実行主体に対して、CPU使用時間を配分することができる。

【0026】また、本発明に係るスケジューリング方法は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング方法であり、各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、優先度の高い第一の実行主体が、優先度の低い第二の実行主体の処理を待つ必要があるときには、当該第一の実行主体に対応するタイムスロットデータを当該第二の実行主体に割り当てるステップと、当該第二の実行主体の処理が終了したときに、当該第一の実行主体から割り当てられたタイムスロットデータを、当該第一の実行主体に再度割り当てるステップとを具備することを特徴とする。

【0027】以上のような本発明に係るスケジューリング方法では、実行主体から独立したタイムスロットデー

タをスケジューリングの対象とすることにより、優先度の高い実行主体が、優先度の低い実行主体の処理を待つ必要のあるときには、タイムスロットデータを受け渡すという低コストな処理によって優先度継承を行うことができる。

【0028】また、本発明に係るスケジューリング方法は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのスケジューリング方法であり、各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、タイムスロットデータに対応する実行主体の優先度に応じて、リングバッファ上の複数の待ち行列から当該タイムスロットデータを配置すべき待ち行列を決定する位置決定ステップと、タイムスロットデータを前記リングバッファ上の待ち行列に配置するときに、当該リングバッファ上の待ち行列の最後尾に当該タイムスロットデータを配置するステップと、所定のリングバッファ上の待ち行列の先頭に位置するタイムスロットデータに処理時間を与えるステップとを具備することを特徴とする。

【0029】以上のような本発明に係るスケジューリング方法では、タイムスロットデータに対応する実行主体の優先度に応じて、当該タイムスロットデータを配置すべきリングバッファ上の位置を決定し、同じ優先度のタイムスロットデータがリングバッファ上で同じ位置を共有することにより、タイムスロットの優先度に応じて、タイムスロットデータに対応する実行主体に対して、CPU使用時間を配分することができる。

【0030】また、本発明に係る記録媒体は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのデータ処理プログラムが記録された記録媒体であって、各実行主体毎に、スケジューリングの対象として割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、優先度の高い第一の実行主体が、優先度の低い第二の実行主体の処理を待つ必要があるときには、当該第一の実行主体に対応するタイムスロットデータを当該第二の実行主体に割り当てるステップと、当該第二の実行主体の処理が終了したときに、当該第一の実行主体から割り当てられたタイムスロットデータを、当該第一の実行主体に再度割り当てるステップとを処理するデータ処理プログラムを提供することを特徴とする。

【0031】また、本発明に係る記録媒体は、複数の実行主体を時分割処理することが可能なオペレーティングシステムのデータ処理プログラムが記録された記録媒体であって、各実行主体毎に、スケジューリングの対象と

して割り当てられたタイムスロットデータに処理時間を割り当て、時分割スケジューリングを行うステップと、処理時間が割り当てられたタイムスロットデータに対応する実行主体を実行するステップと、タイムスロットデータに対応する実行主体の優先度に応じて、リングバッファ上の複数の待ち行列から当該タイムスロットデータを配置すべき待ち行列を決定する位置決定ステップと、タイムスロットデータを前記リングバッファ上の待ち行列の一つに配置するとき、当該リングバッファ上の待ち行列の最後尾に当該タイムスロットデータを配置するステップと、所定のリングバッファ上の待ち行列の先頭に位置するタイムスロットデータに処理時間を与えるステップとを処理するデータ処理プログラムを提供する。

【0032】

【発明の実施の形態】以下、本発明の実施の形態について、図面を参照しながら詳細に説明する。

【0033】本発明では、複数の実行主体を時分割処理することが可能なオペレーティングシステムにおいて、各スレッドに対応するようにタイムスロットを割り当てて、各スレッドに割り当てられたタイムスロットデータを単位としてスケジューリングを行う。すなわち、本発明では、スレッド自体をスケジューリングの対象とするのではなく、各スレッドにタイムスロットデータを割り当てて、それらのタイムスロットデータをスケジューリングの対象とする。そして、タイムスロットデータに処理時間が割り当てられたら、当該タイムスロットデータに対応したスレッドを実行するようにする。

【0034】ここで、タイムスロットデータとは、各スレッドに割り当てられている優先度を間接的に表現したものである。各タイムスロットデータは、当該タイムスロットデータにCPUの処理時間が与えられたときに実行すべきスレッドへのリンク情報を保持する。そして、オペレーティングシステムは、スレッドを単位としてスケジューリングを行うのではなく、タイムスロットデータを単位としてスケジューリングを行い、タイムスロットデータにCPUの処理時間が割り当てられたら、その時点において当該タイムスロットデータに対応しているスレッドを実行する。

【0035】そして、本発明では、タイムスロットデータとスレッドとの対応関係を動的に変更することで、優先度の継承を実現する。すなわち、優先度の高いスレッドが、優先度の低いスレッドの処理を待つ必要がある場合には、当該処理が完了するまでの間、優先度の高いスレッドに対応していたタイムスロットデータを、優先度の低いスレッドに対応したタイムスロットデータとして扱い、当該タイムスロットデータに処理時間が割り当てられたら、優先度の低いスレッドを実行する。

【0036】なお、本発明は、共有資源に対するアクセスの排他制御だけでなく、オブジェクト指向オペレーティングシステムにおける同期ベースのメッセージ通信

や、何らかのイベント待ちなど、複数のスレッド間で依存関係を持つ待ち事象一般に適用することが可能である。

【0037】以下、このような本発明について、本発明を適用したスケジューリング方法の具体的な例を挙げて詳細に説明する。なお、以下に説明するスケジューリング方法は、例えば、複数のスレッドを時分割処理することが可能なオペレーティングシステムにおけるスケジューリング装置において実行される。そして、このようなオペレーティングシステムが記録された記録媒体が、本発明に係る記録媒体に該当する。

【0038】1. スケジューリングに使用するデータ

まず、以下に説明するスケジューリング方法において使用されるデータについて説明する。

【0039】以下に示すスケジューリング方法では、各スレッドに対応するようにタイムスロットデータを割り当てて、各スレッドに割り当てられたタイムスロットデータを単位としてスケジューリングを行う。すなわち、スレッドが生成される毎に、当該スレッドに対応したタイムスロットデータが生成される。そして、このときに、各スレッド毎に、当該スレッドについてのデータ（以下、スレッドデータと称する。）が格納されるデータ領域が、例えば、図24における外部記憶装置10上に確保され、当該データ領域にスレッドデータが格納される。また、タイムスロットデータが格納されるデータ領域が、同じく例えば、図24における外部記憶装置10上に確保される。

【0040】

【表1】

スレッドデータ
ThreadState state Thread* depend ThreadList inherit TimeSlot* timeSlot

【0041】スレッドデータには、表1に示すように、スレッドの状態を示すためのデータ型である「ThreadState」型の変数「state」と、スレッドデータを特定するためのデータ型である「Thread\*」型の変数「depend」と、スレッドデータリストへのエントリを示すデータ型である「ThreadList」型の変数「inherit」と、タイムスロットを特定するためのデータ型である「TimeSlot\*」型の変数「timeSlot」とが含まれる。

【0042】変数「state」には、スレッドの状態を示す値が設定される。スレッドは、スレッドが停止している状態（以下、「休止状態」と称する。）と、スレッドが何らかの事象を待って中断している状態（以下、「待ち状態」と称する。）と、スレッドが実行可能な状態（以下、「実行可能状態」と称する。）との3つの状態をとる。そして、変数「state」は、スレッドがこ

これらの状態のうちのいずれの状態であるかを示す。なお、スレッドが実行されるのは、スレッドが実行可能状態のときだけである。換言すれば、スケジューリングにおいて、タイムスロットに対してCPUの処理時間が与えられるのは、当該タイムスロットに対応するスレッドが実行可能状態のときだけである。

【0043】変数「depend」には、スレッドが他のスレッドが起こす事象を待つ待ち状態になっている場合に、待ち状態の原因となっているスレッドを指し示す値が設定される。なお、その他の場合、変数「depend」には、NULL値が設定される。

【0044】変数「inherit」には、スレッドが他のスレッドから優先度を継承している場合に、優先度の継承元のスレッドを指し示す値が設定される。換言すれば、変数「inherit」には、このスレッドが起こす事象を待つ待ち状態となっているスレッドを指し示す値が設定される。なお、優先度を継承していない場合、変数「inherit」には、NULL値が設定される。

【0045】なお、スレッドが起こす事象を待つ待ち状態となっているスレッドは、複数ある場合があり、それらのスレッドはスレッドリストに登録される。そして、変数「inherit」は、そのようなスレッドリストへのエントリとなっている。すなわち、変数「inherit」には、優先度の継承元のスレッド自体を指し示す値が設定されるのではなく、そのようなスレッドに登録されたスレッドリストを指し示す値が設定される。

【0046】変数「timeSlot」には、スレッドに対して、スレッド初期化時から割り当てられているタイムスロットデータを指し示す値が設定される。換言すれば、変数「timeSlot」は、スレッドに割り当てられたタイムスロットデータを指し示すリンクデータである。

【0047】以上のようなスレッドデータにおいて、変数「depend」及び「inherit」は、優先度継承関係のリンク情報となる。すなわち、例えば、スレッドAが、スレッドBが起こす事象を待つ待ち状態となっている場合には、スレッドAのスレッドデータの変数「depend」と、スレッドBのスレッドデータの変数「inherit」とが、対になって双方向リンクを形成することとなる。なお、以下の説明においては、スレッドのスレッドデータの変数のことを、単にスレッドの変数と呼ぶことにする。また、スレッドデータのことを単にスレッドと呼ぶことにする。

【0048】

【表2】

タイムスロットデータ
Thread* owner Priority prio TimeSlot* next

【0049】タイムスロットデータには、表2に示すように、スレッドを特定するためのデータ型である「Thread\*」型の変数「owner」と、優先度を特定するためのデータ型である「Priority」型の変数「prio」と、タイムスロットデータを特定するためのデータ型である「TimeSlot\*」型の変数「next」とが含まれる。

【0050】変数「owner」には、タイムスロットデータにCPUの処理時間が割り当てられたときに実行すべきスレッドを指し示す値が設定される。換言すれば、変数「owner」は、タイムスロットデータに対応したスレッドを指し示すリンクデータである。

【0051】なお、以下に説明するスケジューリング方法では、変数「owner」が指し示すスレッドを変更することにより、CPUの処理時間が割り当てられるスレッドを変更し、これにより、実質的に優先度の継承を行う。すなわち、タイムスロットデータにCPUの処理時間が割り当てられたら、当該タイムスロットデータの変数「owner」によって示されるスレッドを実行するようにする。そして、優先度の継承は、待ち行列の変更を行うのではなく、変数「owner」を変更することで行う。

【0052】変数「prio」には、優先度を示す値が設定される。以下に説明するスケジューリング方法では、この変数「prio」に基づいて、各タイムスロットデータに割り当てられるCPUの処理時間が決定される。なお、変数「prio」は、変数「owner」が指し示すスレッドが変更となった場合にも変更されない。すなわち、変数「prio」は、変数「owner」が指し示すスレッドが変更されても、タイムスロットデータにもともと対応していたスレッドの優先度を示す値がそのまま保持される。

【0053】変数「next」には、次にCPUの処理時間が割り当てられるタイムスロットデータを指し示す値が設定される。すなわち、あるタイムスロットデータに割り当てられていたCPUの処理時間を使い切ったら、その後、当該タイムスロットデータの変数「next」が指し示すタイムスロットデータにCPUの処理時間が割り当てられる。

【0054】2. スレッドが待ち状態となる場合の処理つぎに、優先度の高いスレッドが、優先度の低いスレッドが起こす事象を待つ待ち状態となる場合に行われる処理の流れについて説明する。

【0055】優先度の高いスレッドが、優先度の低いスレッドが起こす事象を待つ待ち状態となる場合には、関数「makeWait (Thread\* target, Thread\* depend)」が呼ばれる。ここで、関数「makeWait (Thread\* target, Thread\* depend)」の第1の引数「target」は、「Thread\*」型の引数であり、この引数「target」には、優先度の低いスレッドが起こす事象を待つ待ち状態となるスレッドを指し示す値が設定される。また、第2の引数「depend」は、「Thread\*」型の引数であり、この引数「depend」には、優先度の高いスレッドの待ち状態の

原因となった優先度の低いスレッドを指し示す値が設定される。

【0056】以下、関数「makeWait (Thread\* target, Thread\* depend)」が呼ばれた場合の処理について、図1のフローチャートを参照しながら説明する。なお、図1のフローチャートでは、各ステップでの処理をC言語での表記に準じた表現で示している。

【0057】関数「makeWait (Thread\* target, Thread\* depend)」が呼ばれた場合には、まず、ステップST1-1において、「TimeSlot\*」型の変数「timeSlot」に、引数「target」が指し示すスレッドの変数「timeSlot」の値を設定する。

【0058】次に、ステップST1-2において、引数「target」が指し示すスレッドの変数「state」に、当該スレッドが待ち状態であることを示す値「WAIT」を設定する。

【0059】次に、ステップST1-3において、引数「target」が指し示すスレッドの変数「depend」に、引数「depend」の値を設定する。すなわち、優先度の高いスレッドの待ち状態の原因となったスレッドを、変数「depend」によって参照できるようにする。

【0060】次に、ステップST1-4において、引数「depend」が指し示すスレッドの変数「inherit」が指し示すスレッドリストの末尾に、引数「target」が指し示すスレッドを追加する。なお、図1中の「AddLast(target)」は、スレッドリストの末尾にスレッドを追加する処理を行う関数を示している。すなわち、タイムスロットデータの継承元である優先度の高いスレッドのリストをスレッドの変数「inherit」によって参照可能にする。

【0061】次に、ステップST1-5において、引数「target」に、それまで引数「target」が指し示していたスレッドの変数「depend」の値を設定する。すなわち、実行対象となる可能性のあるスレッドを引数「target」によって参照できるようにする。

【0062】次に、ステップST1-6において、引数「target」が指し示すスレッドのスレッドデータの変数「state」に、当該スレッドが待ち状態であることを示す値「WAIT」が設定されているか否かを判別する。そして、スレッドが待ち状態であることを示す値「WAIT」が設定されている場合は、ステップST1-5へ戻って処理を繰り返し、設定されていない場合は、ステップST1-7へ進む。すなわち、待ち状態でないスレッドが見つかるまでステップST1-5とST1-6の操作を繰り返す。

【0063】ステップST1-7では、変数「timeSlot」が指し示すタイムスロットの変数「owner」に、現在の引数「target」の値を設定する。すなわち、この処理によって、タイムスロットデータによって間接的に表現された優先度が、優先度の高いスレッドから優先度の低

いスレッドへと継承される。

【0064】そして、最後にステップST1-8において、現在の引数「target」が指し示すスレッドの実行を開始する。

【0065】以上の処理により、優先度の高いスレッドが待ち状態とされ、優先度の低いスレッドが実行される。このとき、優先度の高いスレッドの変数「depend」と、優先度の低いスレッドの変数「inherit」とによって、実行待ち関係の相互リンクが形成される。

【0066】なお、優先度の高いスレッドの待ち状態の原因となった優先度の低いスレッドが、更に優先度の低いスレッドが起こす事象を待つ待ち状態となっていたときには、ステップST1-5及びステップST1-6の処理の繰り返しにより、引数「target」に更に優先度の低いスレッドを指し示す値が設定され、その結果、更に優先度の低いスレッドが実行されることとなる。換言すれば、関数「makeWait (Thread\* target, Thread\* depend)」を実行することにより、優先度の高いスレッドに割り当てられていたタイムスロットデータの変数「owner」は、唯一実行可能なスレッドに対応するように変更される。

【0067】そして、本例に係るスケジューリング方法では、タイムスロットデータにCPUの処理時間が割り当てられたら、当該タイムスロットデータの変数「owner」によって示されるスレッドを実行するようにする。したがって、以上のような処理によって変数「owner」を変更することにより、優先度の高いスレッドにタイムスロットデータの変数「prio」に応じて割り当てられていたCPUの処理時間が、先に実行すべき優先度の低いスレッドに割り当てられることとなる。この結果、優先度の高いスレッドから優先度の低いスレッドへ、優先度が実質的に継承されることとなる。

### 【0068】3. スレッドの待ち状態が解消された場合の処理

つぎに、優先度の高いスレッドが、優先度の低いスレッドが起こす事象を待つ待ち状態となっていたときに、優先度の低いスレッドでの処理が完了して、優先度の高いスレッドの待ち状態が解消された場合の処理について説明する。

【0069】優先度の高いスレッドが、優先度の低いスレッドが起こす事象を待つ待ち状態となっていたときに、優先度の低いスレッドでの処理が完了して、優先度の高いスレッドの待ち状態が解消された場合には、関数「makeReady (Thread\* target)」が呼ばれる。ここで、関数「makeReady (Thread\* target)」の引数「target」は、「Thread\*」型の引数であり、この引数「target」には、待ち状態が解消されたスレッドを指し示す値が設定される。

【0070】以下、関数「makeReady (Thread\* target)」が呼ばれた場合の処理について、図2のフローチ

チャートを参照しながら説明する。なお、図2のフローチャートでは、各ステップでの処理をC言語での表記に準じた表現で示している。

【0071】関数「makeReady (Thread\* target)」が呼ばれた場合には、まず、ステップST2-1において、引数「target」が指し示すスレッドの変数「state」に、当該スレッドが実行可能状態であることを示す値「RUNNING」を設定する。

【0072】次に、ステップST2-2において、引数「target」が指し示すスレッドのタイムスロットデータを継承しているスレッドの変数「inherit」に繋がれた継承元スレッドへのリストから、関数「Remove(target)」によって、引数「target」で示されるスレッドへのリンクを削除する。すなわち、これによって継承先のスレッドに対して、タイムスロットデータ返還の手続きを遂行する。

【0073】次に、ステップST2-3において、引数「target」が指し示すスレッドの変数「depend」に、NULL値を設定する。すなわち、当該スレッドの待ち状態の原因となっていたスレッドの処理が終了することにより、もはや待ち合わせが必要なくなったことを確かにする。

【0074】次に、ステップST2-4において、関数「makeReady (Thread\* target)」に受け渡されている引数「target」の値を第1の引数及び第2の引数の両方に設定して、関数「changeOwner (Thread\* target, Thread\* depend)」を呼び出す。すなわち、当該スレッドに元来割り当てられているタイムスロットデータを継承先のスレッドから当該スレッドへと移動する処理を行う。これにより、当該スレッドが当該タイムスロットデータを利用できるようにする。なお、関数「changeOwner (Thread\* target, Thread\* depend)」では、タイムスロットデータの変数「owner」を変更する処理が行われるが、この処理については後で詳細に説明する。そして、関数「changeOwner (Thread\* target, Thread\* depend)」での処理が完了したら、ステップST2-5へ進む。

【0075】そして、最後にステップST2-5において、引数「target」が指し示すスレッドの実行を開始する。

【0076】以上のような関数「makeReady (Thread\* target)」での処理により、優先度の高いスレッドが待ち状態から実行可能状態に戻され、優先度の高いスレッドが実行される。

【0077】つぎに、関数「changeOwner (Thread\* target, Thread\* depend)」が呼ばれた場合の処理について、図3のフローチャートを参照しながら説明する。なお、図3のフローチャートでは、各ステップでの処理をC言語での表記に準じた表現で示している。

【0078】なお、関数「changeOwner (Thread\* target,

Thread\* depend)」の第1の引数「target」及び第2の引数「depend」は、「Thread\*」型の引数であり、関数「changeOwner (Thread\* target, Thread\* depend)」は、引数「target」のタイムスロットデータの変数「owner」を引数「depend」の値に変更する。

【0079】関数「changeOwner (Thread\* target, Thread\* depend)」が呼ばれた場合には、まず、ステップST3-1において、「TimeSlot\*」型の変数「timeSlot」に、引数「target」が指し示すスレッドの変数「timeSlot」の値を設定する。

【0080】次に、ステップST3-2において、変数「timeSlot」が指し示すタイムスロットの変数「owner」に、引数「depend」の値を設定する。すなわち、これにより当該タイムスロットの所有権を引数「depend」で示されるスレッドに継承する。

【0081】次に、ステップST3-3において、引数「target」が指し示すスレッドの変数「inherit」が指し示すスレッドリストの先頭のスレッドを指し示す値を、「Thread\*」型の変数「inherit」に設定する。なお、スレッドリストが空の場合、変数「inherit」にはNULL値を設定する。

【0082】なお、図3中の「Snoop()」は、スレッドリストの先頭のスレッドを取り出す処理を行う関数を示している。そして、この関数「Snoop()」は、先頭のスレッドを指し示す値を戻り値として返す。

【0083】次に、ステップST3-4において、変数「inherit」にNULL値が設定されているか否かを判別する。そして、変数「inherit」にNULL値が設定されている場合は、この関数「changeOwner (Thread\* target, Thread\* depend)」の処理を終了する。また、変数「inherit」にNULL値が設定されていない場合は、ステップST3-5へ進む。

【0084】ステップST3-5では、現在の変数「inherit」の値を第1の引数に設定するとともに、現在の引数「depend」の値を第2の引数に設定して、改めて関数「changeOwner (Thread\* inherit, Thread\* depend)」を呼び出す。そして、関数「changeOwner (Thread\* inherit, Thread\* depend)」での処理が完了したら、ステップST3-6へ進む。

【0085】ステップST3-6では、引数「inherit」が指し示す次のスレッドを指し示す値を、変数「inherit」に設定する。なお、次のスレッドが無い場合、変数「inherit」にはNULL値を設定する。そして、ステップST3-6での処理が完了したら、ステップST3-4へ戻って処理を繰り返す。

【0086】以上のような関数「changeOwner (Thread\* target, Thread\* depend)」での処理により、タイムスロットデータの変数「owner」が変更されるとともに、優先度の高いスレッドの変数「depend」と、優先度の低いスレッドの変数「inherit」とによって形成され

ていた実行待ち関係のリンクが解消される。

【0087】なお、優先度の低いスレッドが起こす事象を待ち待ち状態となっているスレッドは複数ある場合があり、それらのスレッドはスレッドリストに登録されている。そこで、関数「changeOwner (Thread\* target, Thread\* depend)」では、ステップST3-4乃至ステップST3-6の処理を繰り返すことより、スレッドリストに登録されている全てのスレッドについて、それらのスレッドに対応するタイムスロットデータの変数「owner」の変更を行うようにしている。

#### 【0088】4. スケジューリングの具体例

つぎに、スケジューリングの具体例を、図4乃至図23を参照して説明する。

【0089】なお、以下に説明する例では、いわゆるタイムスライススケジューリングを適用しており、スレッドの優先度が、各スレッドに割り当てられるCPUの処理時間の割合を示すようにしている。そして、以下に示す例では、優先度によって重み付けされたラウンド・ロビン・スケジューリング・アルゴリズム (Round Robin Scheduling Algorithm) を基本としている。

【0090】以下に説明するスケジューリングを行うにあたっては、まず、各スレッドに対して、優先度に応じたタイムスロットデータが割り当てられる。なお、本例において、優先度は15～0の値をとるものとする。すなわち、各タイムスロットデータの変数「prio」には、当該タイムスロットに対応しているスレッドの優先度に応じて、15～0のうちのいずれかの値が設定される。ここで、15～0の値は、それぞれCPUの使用率の比1:1/2:1/3:・・・:1/16に対応する。

【0091】そして、各スレッドに対応した各タイムスロットデータは、全部で17列の待ち行列からなるリングバッファに入れられ、各待ち行列毎に順にタイムスロットデータが選択される。そして、選択されたタイムスロットデータに対してCPUの処理時間が割り当てられ、当該タイムスロットデータに対応したスレッドが実行される。

【0092】CPUの処理時間が割り当てられたタイムスロットデータは、対応するスレッドが、当該タイムスロットに割り当てられた処理時間を使い切った時点で、当該タイムスロットデータの変数「prio」に設定された優先度に従って、何番目か先の待ち行列に繋がる。ここで、当該タイムスロットデータが次に繋がるリングバッファ中の待ち行列を、優先度によって調整することにより、優先度に応じたCPU使用率の割り当てが実現される。

【0093】なお、ここではラウンド・ロビン・スケジューリング・アルゴリズムを基本としたスケジューリング方法を例に挙げるが、本発明はこれに限られるものではない。すなわち、本発明は、例えば、優先度に基づい

てFCFS (First Come First Serve) を行うようなスケジューリング方法等にも適用可能である。ここで、優先度に基づいてFCFSを行うようなスケジューリング方法としては、例えば、実時間システムにおいてよく用いられる「Rate Monotonic」(John Lehoczky, Lui Sha, and Ye Ding. The Rate Monotonic Scheduling Algorithm. Exact Characterization And Average Case Behavior. Technical report, Department Of Statistics, Carnegie Mellon University, 1987) や「Earliest Deadline First」などのアルゴリズムを適用したスケジューリング方法が挙げられる。

【0094】4-1 優先度の継承が行われなるとき  
優先度が15のスレッドAと、優先度が13のスレッドBと、優先度3のスレッドCとを時分割処理する場合であって、優先度の継承が行われな場合について、図4乃至図13を参照して説明する。なお、図4乃至図13並びに後掲する図14乃至図23において、1から17までの数字は、待ち行列の番号を示している。そして、これらの番号に付けられた丸印は、当該番号の待ち行列が現在処理の対象となっていることを示している。

【0095】スレッドA, B, Cを時分割処理する際は、まず、図4に示すように、1番目の待ち行列にタイムスロットデータA, B, Cが並べられる。すなわち、タイムスロットデータAの変数「next」がタイムスロットデータBを指し、タイムスロットデータBの変数「next」がタイムスロットデータCを指している。なお、ここでは、スレッドAに割り当てられたタイムスロットデータをタイムスロットデータA、スレッドBに割り当てられたタイムスロットデータをタイムスロットデータB、スレッドCに割り当てられたタイムスロットデータをタイムスロットデータCとしている。

【0096】そして、まず、1番目の待ち行列の先頭のタイムスロットデータであるタイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行される。なお、タイムスロットデータAに対応したスレッドとは、タイムスロットデータAの変数「owner」が指し示すスレッドのことである。

【0097】そして、当該タイムスロットデータAに割り当てられていた処理時間を、スレッドAが使い切ったら、タイムスロットデータAは、リングバッファ中で、優先度に対応した分だけ先の待ち行列に繋がる。具体的には、スレッドAの優先度が15なので、図5に示すように、タイムスロットデータAは、(16-15)=1の分だけ先の待ち行列、すなわち2番目の待ち行列に繋がる。

【0098】次に、1番目の待ち行列の次のタイムスロットデータであるタイムスロットデータBにCPUの処理時間が割り当てられ、タイムスロットデータBに対応

したスレッドであるスレッドBが実行される。なお、タイムスロットデータBに対応したスレッドとは、タイムスロットデータBの変数「owner」が指し示すスレッドのことである。

【0099】そして、当該タイムスロットデータBに割り当てられていた処理時間を、スレッドBが使い切ったら、タイムスロットデータBは、リングバッファ中で、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドBの優先度が13なので、図6に示すように、タイムスロットデータBは、 $(16-13)=3$ の分だけ先の待ち行列、すなわち4番目の待ち行列に繋がれる。

【0100】次に、1番目の待ち行列の次のタイムスロットデータであるタイムスロットデータCにCPUの処理時間が割り当てられ、タイムスロットデータCに対応したスレッドであるスレッドCが実行される。なお、タイムスロットデータCに対応したスレッドとは、タイムスロットデータCの変数「owner」が指し示すスレッドのことである。

【0101】そして、当該タイムスロットデータCに割り当てられていたタイムスライスを使い切ったら、タイムスロットデータCは、リングバッファ中で、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドCの優先度が3なので、図7に示すように、タイムスロットデータCは、 $(16-3)=13$ の分だけ先の待ち行列、すなわち14番目の待ち行列に繋がれる。

【0102】以上で1番目の待ち行列に並んでいたタイムスロットデータの処理が終了し、次に、2番目の待ち行列に並んでいるタイムスロットデータの処理が行われる。このとき、2番目の待ち行列の先頭は、タイムスロットデータAとなっているので、当該タイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行される。そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図8に示すように、タイムスロットデータAは、 $(16-15)=1$ の分だけ先の待ち行列、すなわち3番目の待ち行列に繋がれる。

【0103】以上で2番目の待ち行列に並んでいたタイムスロットデータの処理が終了し、次に、3番目の待ち行列に並んでいるタイムスロットデータの処理が行われる。このとき、3番目の待ち行列の先頭は、タイムスロットデータAとなっているので、当該タイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行される。そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロ

ットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図9に示すように、タイムスロットデータAは、

$(16-15)=1$ の分だけ先の待ち行列、すなわち4番目の待ち行列に繋がれる。ここで、4番目の待ち行列には、既にタイムスロットデータBが並んでいるので、タイムスロットデータAは、タイムスロットデータBの後ろに繋がれる。

【0104】以上で3番目の待ち行列に並んでいたタイムスロットデータの処理が終了し、次に、4番目の待ち行列に並んでいるタイムスロットデータの処理が行われる。このとき、4番目の待ち行列の先頭は、タイムスロットデータBとなっているので、先ず、当該タイムスロットデータBにCPUの処理時間が割り当てられ、タイムスロットデータBに対応したスレッドであるスレッドBが実行される。そして、当該タイムスロットデータBに割り当てられていた処理時間を使い切ったら、タイムスロットデータBは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドBの優先度が13なので、図10に示すように、タイムスロットデータBは、 $(16-13)=3$ の分だけ先の待ち行列、すなわち7番目の待ち行列に繋がれる。

【0105】次に、4番目の待ち行列の次のタイムスロットデータであるタイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行される。そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図11に示すように、タイムスロットデータAは、 $(16-15)=1$ の分だけ先の待ち行列、すなわち5番目の待ち行列に繋がれる。

【0106】以下同様に、待ち行列毎にタイムスロットデータに対してCPUの処理時間を割り当てて、当該タイムスロットデータに対応したスレッドの実行を行うとともに、処理時間を使い切ったタイムスロットデータを優先度に対応した分だけ先の待ち行列に繋ぐ処理を行っていく。なお、以上のように処理を進めていくことにより、待ち行列が17番目にまで達した場合、その後は1番目の待ち行列に戻るものとする。

【0107】例えば、図12に示すように、17番目の待ち行列にタイムスロットデータAが繋がれており、当該タイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行された場合、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、 $(16-15)=1$ の分だけ先の

待ち行列に繋がれる。そして、このとき現在の待ち行列は17番目なので、 $(16-15)=1$ の分だけ先の待ち行列は、1番目の待ち行列である。すなわち、タイムスロットデータAは、図13に示すように、1番目の待ち行列に繋がれる。

【0108】以上のような処理を順次行っていくことにより、スレッドAのCPU使用率と、スレッドBのCPU使用率と、スレッドCのCPU使用率との比が、 $1:1/3:1/13$ となる。すなわち、スレッドの対応するタイムスロットデータの優先度に応じて、それぞれのスレッドにCPUの処理時間が割り当てられ、複数のスレッドの時分割処理が実行されることとなる。

【0109】ところで、従来の多くのタイムスライススケジューリングでは、CPU使用率の割合に応じて待ち行列のソートを行うなどの処理が必要であった。これに対して、上記の方法では、単にタイムスロットデータの繋ぎ換えを行うだけで、タイムスライススケジューリングを実現することができる。すなわち、上記の方法を用いることで、非常に少ない計算量にて、タイムスライススケジューリングを実現することができる。

【0110】なお、各処理がある一定時間内に終わらなければシステムが成り立たないような実時間システムに上記の方法を適用するような場合には、スケジューリング待ち行列に入れるタイムスロットデータの量を制限するようにすればよい。スケジューリング待ち行列に入れるタイムスロットデータの量を制限することで、CPUの使用率を保証することが可能となり、これにより、実時間システムへの適用が可能となる。

【0111】例えば、スレッドAが最も優先順位の高いスレッドだと仮定する。ここでスレッドAの実行終了には、CPU処理時間を2単位必要とするとする。また、スレッドAは、5CPU単位処理時間以内に終了しなければならないとする。この場合、スレッドAが図8の状態にあって、スレッドAが実行終了までにあと1CPU単位処理時間となった状態で図9のような状態になるとする。このとき、少なくとも次にスレッドAに処理時間が割り当てられるまでには1CPU単位処理時間待たなければならない（すなわちスレッドBが実行する処理時間が、次のサイクルでスレッドAは無事に実行を終了することができる。ところが、仮に、図8の状態で現在スレッドBが繋がれている待ち行列に5個のスレッドが繋がっていると考えると、スレッドAに次に処理時間が回ってくるまでには、5CPU単位処理時間待つことになる。この場合、これだけでスレッドAの実行制限時間を超えてしまう。このようなときに、例えば、一つの待ち行列に繋ぐことのできるスレッドの数を、待ち行列理論などによりスレッドAが制限時間を超えることのないことが示されるような数以内に制限することが考えられる。

#### 【0112】4-2 優先度の継承が行われるとき

優先度が15のスレッドAと、優先度が13のスレッドBと、優先度3のスレッドCとを時分割処理する場合であって、優先度の継承が行われる場合について、図14乃至図23を参照して説明する。

【0113】なお、図14乃至図23において、括弧内の符号は、タイムスロットデータの変数「owner」が指し示しているスレッドを表している。すなわち、例えば、 $A(A)$ は、タイムスロットデータAの変数「owner」がスレッドAを指し示しているということを表しており、また、例えば、 $A(B)$ は、タイムスロットデータAの変数「owner」がスレッドBを指し示しているということを表している。

【0114】本例においても、図4乃至図13に示した例と同様、先ず、図14に示すように、1番目の待ち行列にタイムスロットデータA、B、Cが並べられる。そして、先ず、1番目の待ち行列の先頭のタイムスロットデータであるタイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行されるが、本例では、このときに、スレッドAが、スレッドBが起こす事象を待たねばならないとする。すると、ここではスレッドAは実行されずに、図1に示したフローチャートに従って処理が行われる。その結果、スレッドAが待ち状態とされるとともに、図15に示すように、タイムスロットデータAの変数「owner」がスレッドBを指し示すように変更され、スレッドBが実行される。すなわち、タイムスロットデータAからのリンクが、スレッドAからスレッドBに変更され、もともとスレッドAに対して割り当てられていたCPUの処理時間が、スレッドBに割り当てられることとなる。

【0115】そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図16に示すように、タイムスロットデータAは、 $(16-15)=1$ の分だけ先の待ち行列、すなわち2番目の待ち行列に繋がれる。

【0116】次に、1番目の待ち行列の次のタイムスロットデータであるタイムスロットデータBにCPUの処理時間が割り当てられ、タイムスロットデータBに対応したスレッドであるスレッドBが実行される。なお、タイムスロットデータBに対応したスレッドとは、タイムスロットデータBの変数「owner」が指し示すスレッドのことであり、ここではスレッドBである。

【0117】そして、当該タイムスロットデータBに割り当てられていた処理時間を使い切ったら、タイムスロットデータBは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドBの優先度が13なので、図17に示すように、タイムスロットデータBは、 $(16-13)=3$ の分だけ先の待ち行列、すなわ

ち4番目の待ち行列に繋がれる。

【0118】次に、1番目の待ち行列の次のタイムスロットデータであるタイムスロットデータCにCPUの処理時間が割り当てられ、タイムスロットデータCに対応したスレッドであるスレッドCが実行される。なお、タイムスロットデータCに対応したスレッドとは、タイムスロットデータCの変数「owner」が指し示すスレッドのことであり、ここではスレッドCである。

【0119】そして、当該タイムスロットデータCに割り当てられていた処理時間を使い切ったら、タイムスロットデータCは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドCの優先度が3なので、図18に示すように、タイムスロットデータCは、 $(16-3)=3$ の分だけ先の待ち行列、すなわち14番目の待ち行列に繋がれる。

【0120】以上で1番目の待ち行列に並んでいたタイムスロットデータの処理が終了し、次に、2番目の待ち行列に並んでいるタイムスロットデータの処理が行われる。このとき、2番目の待ち行列の先頭は、タイムスロットデータAとなっているので、当該タイムスロットデータAの変数「owner」が指し示すスレッド、すなわちスレッドBが実行される。そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図19に示すように、タイムスロットデータAは、 $(16-15)=1$ の分だけ先の待ち行列、すなわち3番目の待ち行列に繋がれる。

【0121】以上で2番目の待ち行列に並んでいたタイムスロットデータの処理が終了し、次に、3番目の待ち行列に並んでいるタイムスロットデータの処理が行われる。このとき、3番目の待ち行列の先頭は、タイムスロットデータAとなっているので、当該タイムスロットデータAの変数「owner」が指し示すスレッド、すなわちスレッドBが実行される。そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図20に示すように、タイムスロットデータAは、 $(16-15)=1$ の分だけ先の待ち行列、すなわち4番目の待ち行列に繋がれる。ここで、4番目の待ち行列には、既にタイムスロットデータBが並んでいるので、タイムスロットデータAは、タイムスロットデータBの後ろに繋がれる。

【0122】以上で3番目の待ち行列に並んでいたタイムスロットデータの処理が終了し、次に、4番目の待ち行列に並んでいるタイムスロットデータの処理が行われる。このとき、4番目の待ち行列の先頭は、タイムスロットデータBとなっているので、先ず、当該タイムスロットデータBの変数「owner」が指し示すスレッド、す

なわちスレッドBが実行される。

【0123】このときに、スレッドBの実行により、スレッドAの待ち状態の原因が解消されたとする。すると、図2及び図3に示したフローチャートに従って処理が行われ、その結果、スレッドAが実行可能状態にされるとともに、図21に示すように、タイムスロットデータAの変数「owner」がスレッドAを指し示すように変更される。すなわち、タイムスロットデータAからのリンクが、スレッドAに戻される。

【0124】その後、タイムスロットデータBに割り当てられていた処理時間を使い切ったら、タイムスロットデータBは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドBの優先度が13なので、図22に示すように、タイムスロットデータBは、 $(16-13)=3$ の分だけ先の待ち行列、すなわち7番目の待ち行列に繋がれる。

【0125】次に、4番目の待ち行列の次のタイムスロットデータであるタイムスロットデータAにCPUの処理時間が割り当てられ、タイムスロットデータAに対応したスレッドであるスレッドAが実行される。なお、タイムスロットデータAに対応したスレッドとは、タイムスロットデータAの変数「owner」が指し示すスレッドのことであり、ここではスレッドAとなっている。

【0126】そして、当該タイムスロットデータAに割り当てられていた処理時間を使い切ったら、タイムスロットデータAは、優先度に対応した分だけ先の待ち行列に繋がれる。具体的には、スレッドAの優先度が15なので、図23に示すように、タイムスロットデータAは、 $(16-15)=1$ の分だけ先の待ち行列、すなわち5番目の待ち行列に繋がれる。

【0127】以下同様に、待ち行列毎にタイムスロットデータの実行を行うとともに、処理時間を使い切ったタイムスロットデータを優先度に対応した分だけ先の待ち行列に繋ぐ処理を行っていく。

#### 【0128】5. まとめ

上記スケジューリング方法では、優先度の高いスレッドが優先度の低いスレッドの処理を待つ必要があるときに、優先度の高いスレッドの状態を待ち状態にするとともに、優先度の高いスレッドに割り当てられていたタイムスロットデータの変数「owner」が指し示すスレッドを、優先度の高いスレッドから優先度の低いスレッドに変更するようにしている。すなわち、優先度の高いスレッドに割り当てられていたタイムスロットデータからスレッドへのリンクを、一時的に優先度の低いスレッドに切り換えるようにしている。

【0129】換言すれば、上記スケジューリング方法では、優先度の高いスレッドが優先度の低いスレッドの処理を待つ必要があるときに、優先度の低いスレッドの優先度を上げることにより優先度逆転問題を回避するのではなく、優先度の高いスレッドの代わりに優先度の低い

スレッドを動かすようにすることで優先度逆転問題を回避している。

【0130】このようなスケジューリング方法によれば、優先度の高いスレッドが優先度の低いスレッドの処理を待つ必要があるときに、優先度継承のために待ち行列を変更するような処理や優先度を再計算するような処理等を一切行わずに、優先度の高いスレッドから優先度の低いスレッドに優先度を継承したのと同じ結果を得ることができる。したがって、上記スケジューリング方法によれば、特に実時間システムにおいては致命的な問題となる優先度逆転問題を回避することができる。しかも、上記スケジューリング方法では、優先度継承のために待ち行列を変更するような処理や優先度を再計算するような処理等を一切行う必要がないので、スケジューリングコストを増大させることなく、優先度逆転問題を回避することができる。さらに、上記スケジューリング方法では、待ち状態となるスレッドに割り当てられていたCPUの処理時間は、待ち状態の原因となっているスレッドに割り当てられることとなるので、CPUの処理時間は無駄なく有効に利用される。

【0131】なお、上記スケジューリング方法では、待ち事象が連結するような場合（例えば、スレッドAがスレッドBを待ち、スレッドBがスレッドCを待つような場合）にも、全タイムスロットデータが連結の先端のスレッドにリンクされることとなる。したがって、CPUの処理時間が全て有効に使用される。すなわち、上記スケジューリング方法では、従来の優先度継承方法では不可能であった、いわゆる複数スレッド待ちが可能となっている。

【0132】最後に、本発明が適用されるハードウェア構成及びオペレーティングシステム構成の一例について、図24及び図25を参照しながら説明する。なお、図24においては、本発明を適用したテレビジョン受信装置1を例に挙げるが、本発明は他の各種情報処理装置に対して適用することができることは言うまでもない。すなわち、本発明は、時分割処理するマルチスレッドシステムが採用されるオペレーティングシステム（以下、OSという）が搭載される情報処理装置に対して広く適用可能であり、例えば、各種のオーディオ・ビジュアル機器（いわゆるAV機器）、事務用情報処理機器、コンピュータ機器等の情報処理装置に対して適用可能である。

【0133】図24において、テレビジョン受信装置1は、アンテナ又はケーブル等によって放送局から送信された信号を受信し、受信した信号に基づいて、画像表示装置に映像を表示するとともに、スピーカから音声出力する装置である。

【0134】テレビジョン受信装置1は、一般的なテレビジョン受信機能を備えているだけでなく、外部からプログラムやデータ等を受け取ることが可能とされてお

り、図24に示すように、バス/I Oブリッジ2を介してバス3に接続されたテレビ機能部4と、バス/メモリブリッジ5を介してバス3に接続されたプロセッサ6と、バス/メモリブリッジ5を介してバス3に接続されたROM（Read Only Memory）7及びRAM（Random Access Memory）8と、バス3に接続された操作パネル9、外部記憶装置10及び通信装置11とを備えている。

【0135】テレビ機能部4は、アンテナ又はケーブル等によって受信した信号に基づいて、映像や音声を再生する機能を備えている。このテレビ機能部4は、バス/I Oブリッジ2を介してバス3に接続されており、これにより、他の各部と信号のやりとりが可能とされている。

【0136】プロセッサ6は、このテレビジョン受信装置1の各部の制御を行うものであり、バス/メモリブリッジ5を介してバス3に接続されている。また、プロセッサ6には、バス/メモリブリッジ5を介してROM7及びRAM8が接続されている。ROM7は、例えば、テレビジョン受信装置1に関する不変の情報を保持している。RAM8は、プロセッサ6のワークエリアとして使われるメモリ空間を提供する。すなわち、プロセッサ6は、後述するように外部記憶装置10に記録されているOS（本発明のスケジューリング装置、方法を含む）やアプリケーションプログラムを、RAM8をワークエリアとして使用しながら実行することにより、このテレビジョン受信装置1を構成する各部を制御する。

【0137】操作パネル9は、ユーザからの操作入力を受け付けるための入力装置である。テレビジョン受信装置1は、この操作パネル9から、例えば、チャンネルやボリューム等の切り替えを指示する信号が入力される。この操作パネル9は、具体的には、各種信号を入力するための複数のボタンを備えた入力装置や、いわゆるマウスに代表されるようなポインティングデバイス等によって構成される。この操作パネル9によって入力された信号は、バス3及びバス/メモリブリッジ5を介してプロセッサ6に入力される。そして、プロセッサ6は、操作パネル9から入力された信号に基づいて、所定の演算処理を行い、テレビジョン受信装置1を構成する各部を制御する。

【0138】外部記憶装置10は、例えばハードディスク装置等の記憶装置から構成され、プロセッサ6による各種制御を行うためのOS（本発明のスケジューリング装置、方法を含む）やアプリケーションプログラム等を記録している。また、外部記憶装置10は、画像データ及び制御データや、外部から通信装置11を介してダウンロードされたプログラム等が記録される。また、通信装置11は、外部との間でデータ通信を行うための入出力部であり、例えばモデムやターミナルアダプター等の通信装置によって構成される。

【0139】また、テレビジョン受信装置1は、外部記憶装置10に記憶されているOSをプロセッサ6によって実行し、このOS上で、ROM7に記録されている制御情報等を用いて、外部記憶装置10に記録されたアプリケーションプログラムを実行することにより、各部の制御を行う。すなわち、このテレビジョン受信装置1は、OSを構成する各種データ処理プログラム（本発明のスケジューリング装置、方法を含む）を提供するプログラム提供媒体として、外部記憶装置10を備えている。

【0140】なお、本発明のスケジューリング装置、方法を含むOSは、ROM7やRAM8に記録しておくとしてもよい。この場合には、ROM7やRAM8がプログラム提供媒体となる。ただし、バージョンアップを行う等のようなOSの書き換え操作を行うためには、書き換え可能な記録媒体に本発明のスケジューリング装置、方法を含むOSを備えることが望ましい。また、プログラム提供媒体としては、例えば、このテレビジョン受信装置1に対して着脱自在に用いられる磁気記録媒体や光記録媒体等であってもよいし、このテレビジョン受信装置1と他の各種情報処理装置とを接続するネットワーク回線等であってもよい。

【0141】テレビジョン受信装置1に搭載されるOSは、純オブジェクト指向OSである。そして、このOS上で、例えば、テレビ機能部4に動画像を表示するためにアプリケーションプログラムや、操作パネル9を制御するためのグラフィカル・ユーザ・インタフェース（GUI）を実現するアプリケーションプログラムが実行される。

【0142】次に、図25を用いてOSの一例について説明する。このOSは、基本部分であるメタコア（Meta Core）20と、その他のオブジェクト群とから構成される。ここで、メタコア20は、オブジェクトとしては定義できない部分であり、オブジェクト間の実行制御の切り替えをする処理部、すなわち本発明に係る処理時間を割り当てられたスレッドを実行する処理部などを含んでいる。

【0143】このOSは、純オブジェクト指向OSであり、OS上で実行されるアプリケーションプログラムを構成するオブジェクトと、OSを構成するオブジェクトとが、同様な実行機構を有するOSである。このOSでは、全てのオブジェクトは、各々一つずつのスレッドを持ち、互いに並行に動作することが可能である。各オブジェクトはメッセージを送信することにより互いに通信することができる。ただし、システムオブジェクトと呼ばれる特定のオブジェクトだけは、メッセージを介することなく通信することができ、さらには本発明にあるようなスケジューリングの対象外にあって動作することが保証されている。また、システムオブジェクトは、特定のデータ構造に対して直接操作することを許可されてい

る。このような特定のデータ構造としては、本発明にあるようなスレッドデータやタイムスロットデータ等が含まれる。

【0144】図25においては、メタコア20以外に、システムオブジェクトであるタイマオブジェクト21とスケジューラ22、及びそれ以外のオブジェクト群として第一のメッセージハンドラ23、第二のメッセージハンドラ24、第一の実行環境マネージャ25、第二の実行環境マネージャ26を示す。なお、図25には実際のシステムに含まれるが省略されている多くのオブジェクト（システムオブジェクトを含む）があることは言うまでもない。

【0145】本OSにおいては、各アプリケーションを異なる実行環境の上で動作させることが可能である。本実施例においては、2つのアプリケーションがそれぞれ異なる実行環境を利用している例が示されている。具体的には、第一のアプリケーション32は、第一の実行環境30の上で動作し、第一の実行環境30は、タイマオブジェクト21、スケジューラ22、第一のメッセージハンドラ23、第一の実行環境マネージャ25を含んでいる。また、第二のアプリケーション33は、第二の実行環境31の上で動作し、第二の実行環境31は、タイマオブジェクト21、スケジューラ22、第二のメッセージハンドラ24、第二の実行環境マネージャ26を含んでいる。また、第一のアプリケーション32及び第二のアプリケーション33も、各々オブジェクト群によって構成されている。さらに、第一のアプリケーションは、第一のオブジェクト34と第二のオブジェクト35を含んでおり、第一のオブジェクト34の優先度は第二のオブジェクト35の優先度より低いと仮定する。

【0146】まず、本発明においてスレッドの切り替えがなされる手順について説明する。第一のアプリケーション32を構成する第一のオブジェクト34が、第二のオブジェクト35に対して処理要求メッセージを送ったとする。この場合、メッセージは、第一のメッセージハンドラ23を経由して、第二のオブジェクト35に送られることになる。このとき、第一のメッセージハンドラ23は、送り手である第一のオブジェクト34の優先度と、受け手である第二のオブジェクト35の優先度を比較することにより、優先度逆転が発生することを検出する。この検出を受けて、第一のメッセージハンドラ23は、関数「makeWait（Thread\* target, Thread\* depend）」を呼び出す。ここで、引数「target」は第一のオブジェクト34のスレッドを示し、引数「depend」は第二のオブジェクト35のスレッドを示す。これによって、第二のオブジェクト35のスレッドに対して、第一のオブジェクト34のスレッドに対応するタイムスロットデータが継承される。

【0147】次に第二のオブジェクト35が第一のオブジェクト34からの処理要求メッセージによって要求さ

れた処理を終了して、第一のオブジェクト34に対して返答メッセージを返すことになる。この返答メッセージは再び第一のメッセージハンドラ23を経由して送られる。このとき、第一のメッセージハンドラ23は、第一のオブジェクト34が第二のオブジェクト35に依存して待ち状態になっていることを検出する。この検出を受けて、第一のメッセージハンドラ23は、関数「makeReady (Thread\* target)」を呼び出す。ここで、引数「t target」は第一のオブジェクト34のスレッドである。これによって、第二のオブジェクト35のスレッドに対して継承されていたタイムスロットデータが、第一のオブジェクト34のスレッドに対して返還される。

【0148】次に時分割スケジューリングについて説明する。時分割スケジューリングにおいては、タイマオブジェクト21が一定時間ごとにタイマ割り込みを起こし、スケジューラ22を呼び出す。スケジューラ22は、既に説明されたような方法で、次に実行されるスレッドのタイムスロットデータを選択し、当該選択されたオブジェクトのスレッドを起動する。

【0149】

【発明の効果】以上詳細に説明したように、本発明では、優先度の高いスレッドに対応していたタイムスロットデータを、優先度の低いスレッドに対応したタイムスロットデータとして扱うようにすることで、優先度の高いスレッドから優先度が低いスレッドへの優先度の継承を実現している。したがって、本発明によれば、優先度逆転の問題を回避することができる。

【0150】しかも、本発明では、優先度の高いスレッドに対応していたタイムスロットデータを優先度の低いスレッドに対応したタイムスロットデータとして扱うようにすることだけで、優先度の継承が実現されるので、優先度の継承を行うにあたって、待ち行列の変更を頻繁に行うような必要はない。したがって、本発明によれば、オーバーヘッドの増加の原因となる待ち行列の変更を頻繁に行うようなことなく、優先度の継承を実現することができる。

【0151】さらに、本発明では、優先度の高いスレッドから優先度の低いスレッドに優先度の継承が行われた場合、優先度の低いスレッドは、優先度の高いスレッドに割り当てられていた処理時間と、優先度の低いスレッドにもともと割り当てられていた処理時間との両方を使うこととなる。したがって、本発明によれば、各スレッドに割り当てられた処理時間を有効に利用することが可能となる。

【図面の簡単な説明】

【図1】スレッドが待ち状態となる場合に呼び出される関数「makeWait (Thread\* target, Thread\* depend)」の処理の流れを示すフローチャートである。

【図2】スレッドの待ち状態が解消された場合に呼び出される関数「makeReady (Thread\* target)」の処理の

流れを示すフローチャートである。

【図3】関数「makeReady (Thread\* target)」の中で呼ばれる関数「changeOwner (Thread\* target, Thread\* depend)」の処理の流れを示すフローチャートである。

【図4】図4乃至図11は優先度の継承が行われない場合の待ち行列の遷移を順に示す図であり、図4は1番目の待ち行列にタイムスロットデータA, B, Cが並んでいる状態を示す図である。

【図5】タイムスロットデータA, B, Cの待ち行列について、図4の次の状態を示す図である。

【図6】タイムスロットデータA, B, Cの待ち行列について、図5の次の状態を示す図である。

【図7】タイムスロットデータA, B, Cの待ち行列について、図6の次の状態を示す図である。

【図8】タイムスロットデータA, B, Cの待ち行列について、図7の次の状態を示す図である。

【図9】タイムスロットデータA, B, Cの待ち行列について、図8の次の状態を示す図である。

【図10】タイムスロットデータA, B, Cの待ち行列について、図9の次の状態を示す図である。

【図11】タイムスロットデータA, B, Cの待ち行列について、図10の次の状態を示す図である。

【図12】図12及び図13は、待ち行列が17番目にまで達した場合の待ち行列の遷移を順に示す図であり、図12は17番目に待ち行列にタイムスロットデータAが並んでいる状態を示す図である。

【図13】タイムスロットデータA, B, Cの待ち行列について、図12の次の状態を示す図である。

【図14】図14乃至図23は優先度の継承が行われる場合の待ち行列の遷移を順に示す図であり、図14は1番目の待ち行列にタイムスロットデータA, B, Cが並んでいる状態を示す図である。

【図15】タイムスロットデータA, B, Cの待ち行列について、図14の次の状態を示す図である。

【図16】タイムスロットデータA, B, Cの待ち行列について、図15の次の状態を示す図である。

【図17】タイムスロットデータA, B, Cの待ち行列について、図16の次の状態を示す図である。

【図18】タイムスロットデータA, B, Cの待ち行列について、図17の次の状態を示す図である。

【図19】タイムスロットデータA, B, Cの待ち行列について、図18の次の状態を示す図である。

【図20】タイムスロットデータA, B, Cの待ち行列について、図19の次の状態を示す図である。

【図21】タイムスロットデータA, B, Cの待ち行列について、図20の次の状態を示す図である。

【図22】タイムスロットデータA, B, Cの待ち行列について、図21の次の状態を示す図である。

【図23】タイムスロットデータA, B, Cの待ち行列について、図22の次の状態を示す図である。

【図24】本発明に係る実施例として示すテレビジョン受信装置の概略構成図である。

【図25】本発明に係るオペレーティングシステムの概略を説明するための図である。

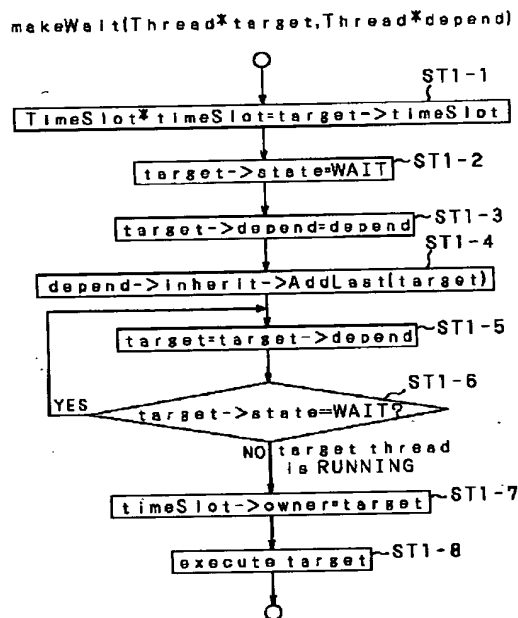
【図26】優先度逆転の問題を説明するための図である。

【図27】従来の優先度継承機構を説明するための図である。

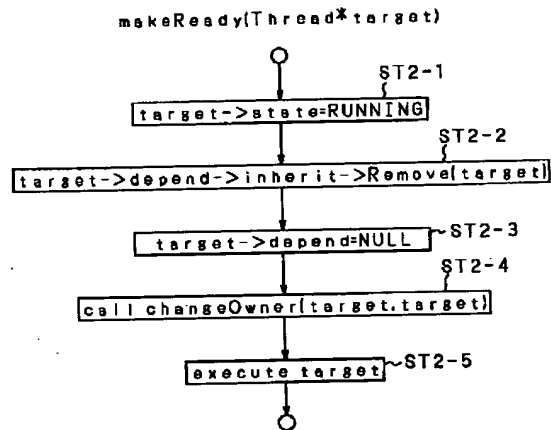
# 【符号の説明】

1 テレビジョン受信装置、 2 バス/I Oブリッジ、 3 バス、 4 テレビ機能部、 5 バス/メモリブリッジ、 6 プロセッサ、 7 ROM (Read Only Memory)、 8 RAM (Random Access Memory)、 9 操作パネル、 10 外部記憶装置、 11 通信装置

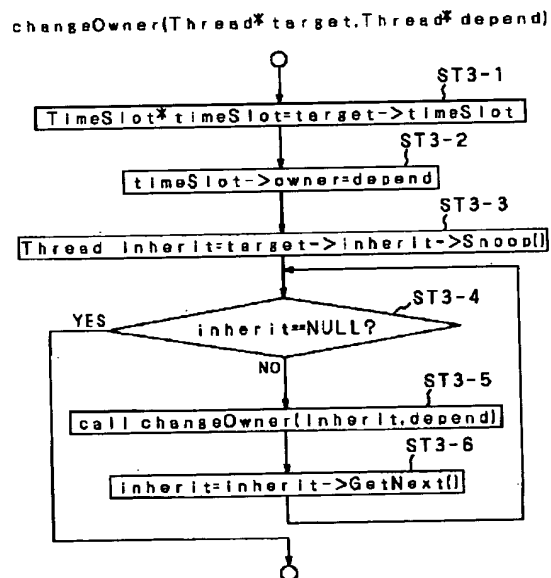
【図1】



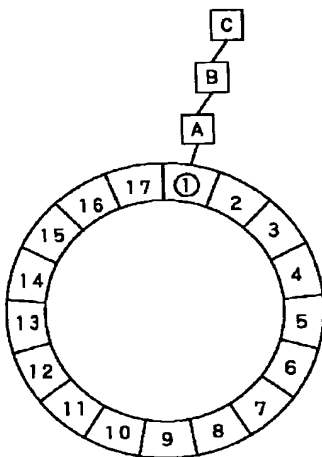
【図2】



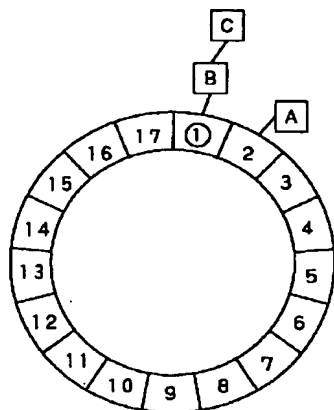
【図3】



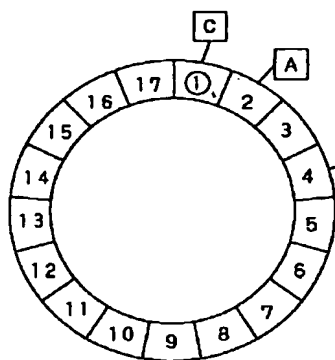
【図4】



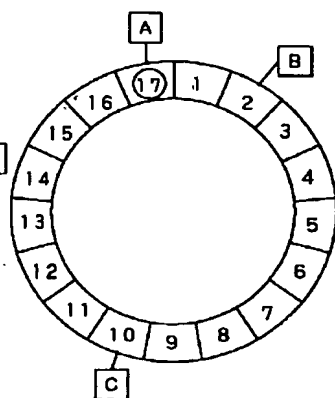
【図5】



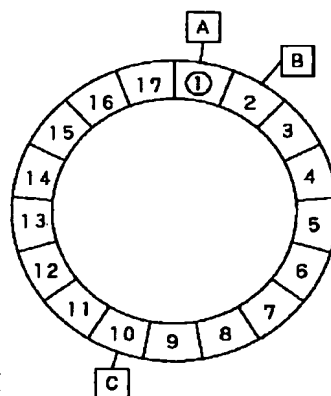
【図6】



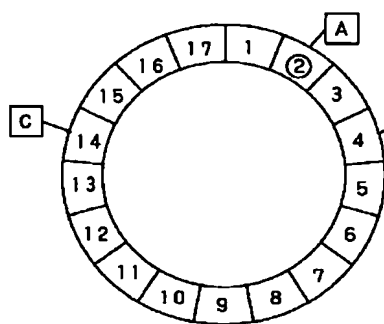
【図12】



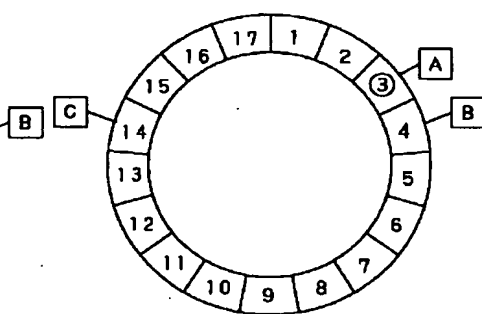
【図13】



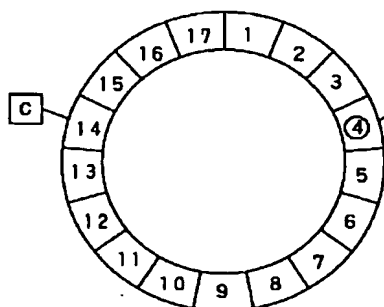
【図7】



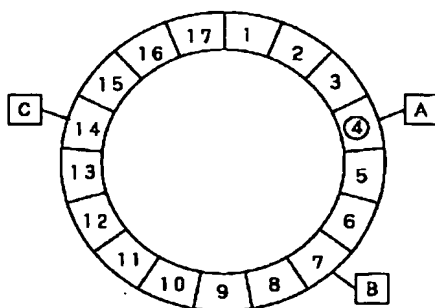
【図8】



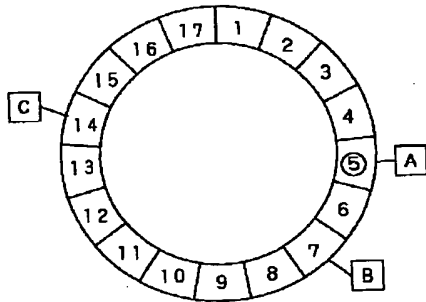
【図9】



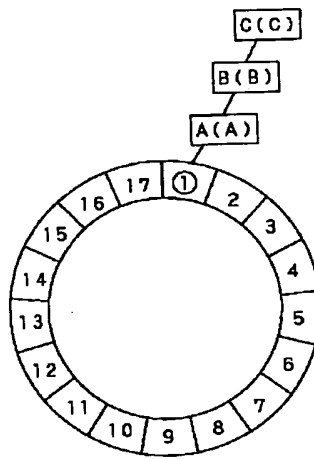
【図10】



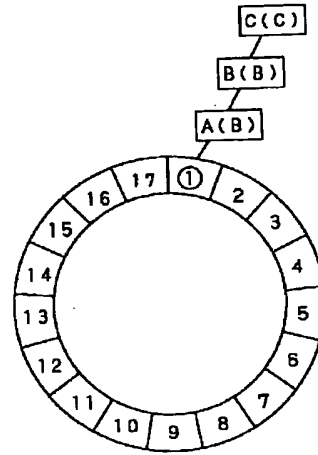
【図11】



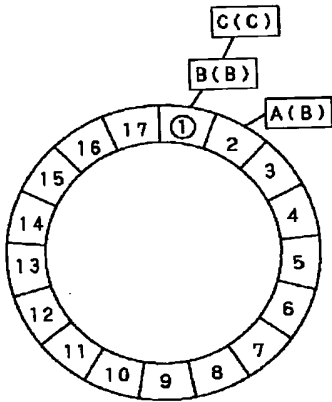
【図14】



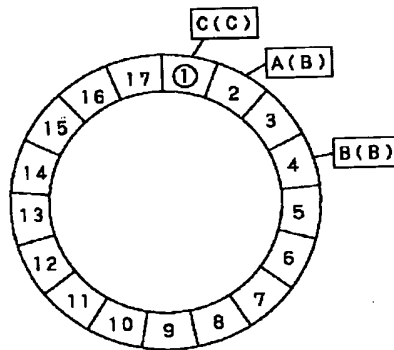
【図15】



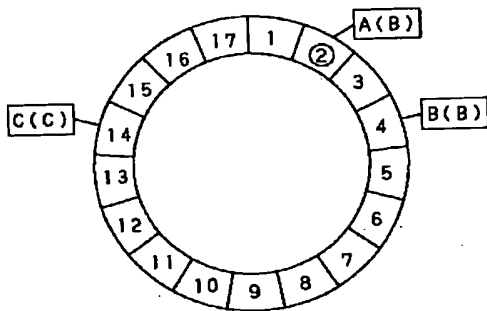
【図16】



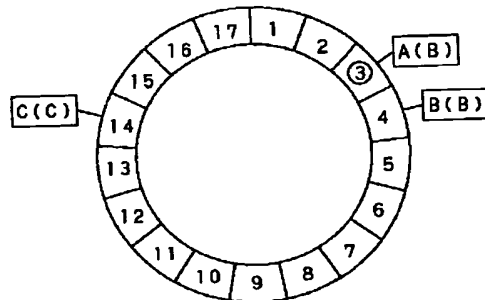
【図17】



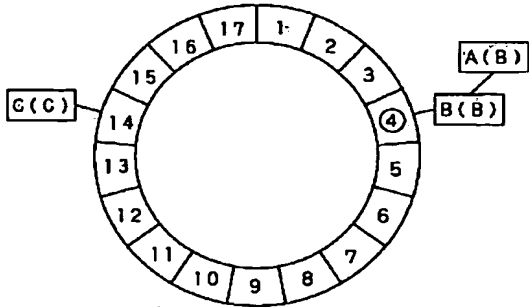
【図18】



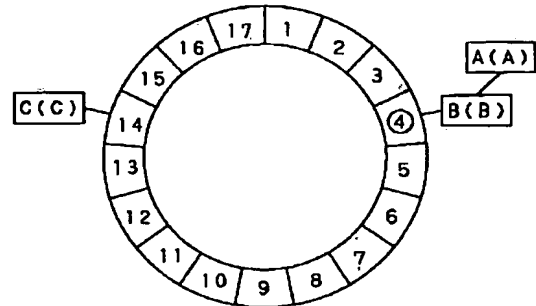
【図19】



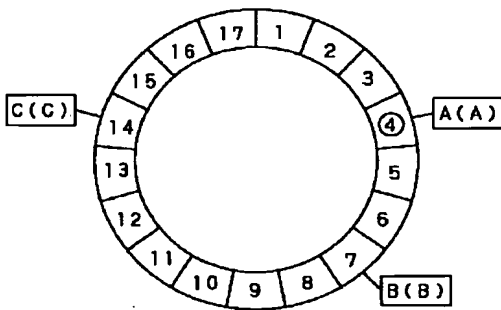
【図20】



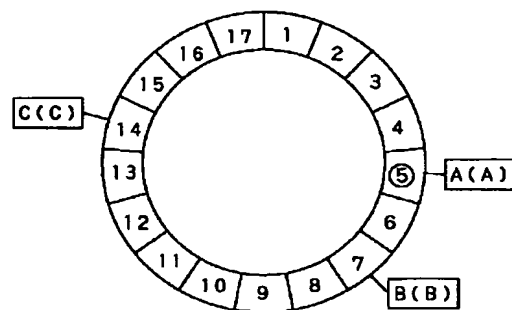
【図21】



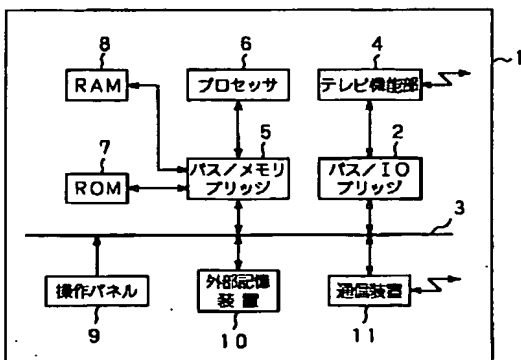
【図22】



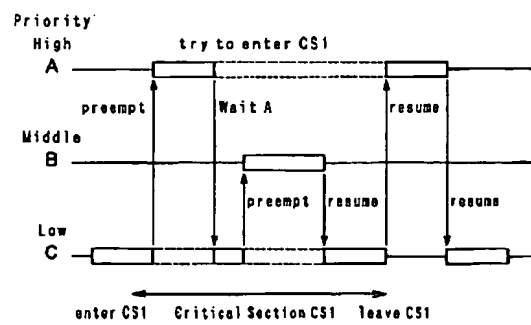
【図23】



【図24】

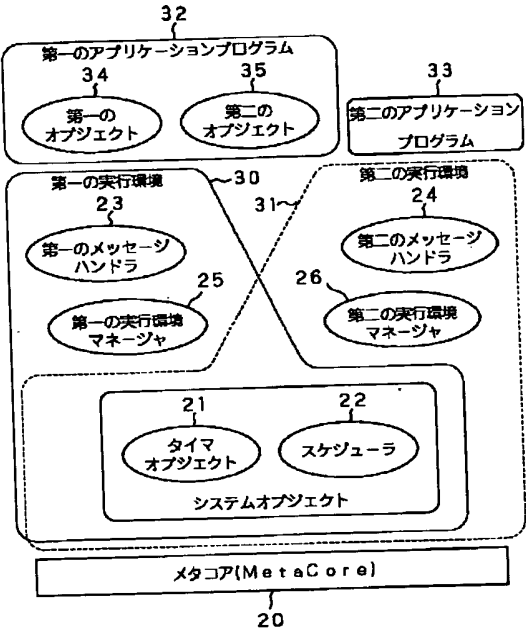


【図26】



優先度逆転の例

【図25】



【図27】

